Learning to Extract Form Labels

Hoa Nguyen
University of Utah
Salt Lake City, UT, USA
thanhhoa@cs.utah.edu

Thanh Nguyen
University of Utah
Salt Lake City, UT, USA
thanhnh@cs.utah.edu

Juliana Freire
University of Utah
Salt Lake City, UT, USA
juliana@cs.utah.edu

ABSTRACT

In this paper we describe a new approach to extract element labels from Web form interfaces. Having these labels is a requirement for several techniques that attempt to retrieve and integrate information that is hidden behind form interfaces, such as hidden Web crawlers and metasearchers. However, given the wide variation in form layout, even within a well-defined domain, automatically extracting these labels is a challenging problem. Whereas previous approaches to this problem have relied on heuristics and manually specified extraction rules, our technique makes use of a learning classifier ensemble to identify element-label mappings; and it applies a reconciliation step which leverages the classifier-derived mappings to boost extraction accuracy. We present a detailed experimental evaluation using over three thousand Web forms. Our results show that our approach is effective: it obtains significantly higher accuracy and is more robust to variability in form layout than previous label extraction techniques.

1. INTRODUCTION

The problem of retrieving and integrating information available in online databases and services has received a lot of attention in the research and industrial communities [7, 22, 1, 6]. This interest is driven both by the quality of the information and the growing number of online databases—it is estimated that there are several million online databases [12].

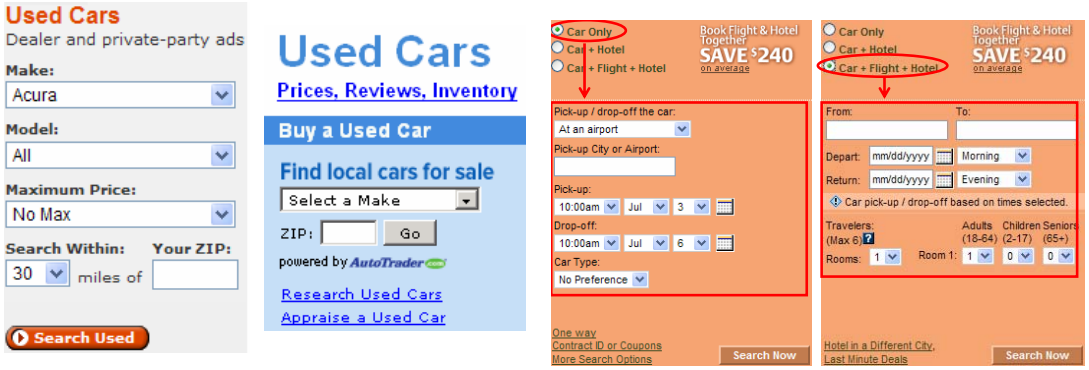
Since most online databases and services can only be accessed through Web form interfaces, to automatically integrate them and retrieve their contents, their form interfaces must be understood [23, 5, 20, 21]. In particular, it is necessary to identify descriptive labels for the form elements. For example, to integrate multiple online databases, these labels are useful for deriving correspondences among the elements in different forms [22, 8]. Labels are also needed for applications that automatically retrieve data hidden behind the form interfaces (*e.g.,* Web integration systems and hidden Web crawlers), which have to derive valid attribute-value assignments to siphon the database contents [19, 16].

In this paper, we propose a new learning-based approach for automatically parsing and extracting element labels of form interfaces that are designed primarily for human consumption. Although the HTML standard provides a `label` attribute to associate descriptive information with individual form elements, it is not widely used. It is common practice to intersperse text representing attribute names with the HTML markup. As Figure 1 illustrates, several different layouts are possible. Labels can be placed in many different positions: on top, in the bottom, to the left, to the right of, and even inside a form element. For example, in Figure 1(a), while the label `Make` is above its corresponding selection list in the form on the left, in the form on the right it is actually one of the values inside the selection list. Figure 1(b) shows a dynamic form whose content and layout change based on which radio button is selected: if the `Car only` button is selected, all labels are placed on top of the form elements. In contrast, if the `Car + Flight + Travel` button is selected, some of the labels are placed to the left of the form elements. Furthermore, a given label may be associated with multiple form elements. The label `Depart` in Figure 1(b) (left), for example, is associated both with the text field for inputting the date and with the selection list for specifying the time. Last, but not least, labels may also be placed outside the form tags in the HTML markup. This wide variation in form design layouts and in nesting relationship between form elements and labels makes the problem of automatically identifying element-label mappings particularly challenging.

Based on the assumption that there are common patterns for label placement, previous approaches to label extraction relied either on heuristics (*e.g.,* based on the textual layout of the page) to *guess* the appropriate label for a given form attribute [10, 16] or on manually-specified extraction rules [23]. These approaches, however, require substantial human input: extraction rules must be manually crafted and maintained. Because of the cost involved in creating these rules, they are often generic and target forms from multiple domains. However, as we discuss later, due to variability of form layout across form domains, higher extraction accuracy can be obtained by designing rules that are specific for a given domain. Another limitation of these approaches is that they only deal with static forms. They cannot handle a significant number of forms that rely on client-side scripting (*e.g.,* JavaScript) to dynamically render their elements.

Contributions and Outline. In this paper, we address a key problem in form interface understanding: how to correctly determine labels associated with elements in both static and dynamic forms. Given a Web form, our goal is to determine a mapping between the form elements and their corresponding textual descriptions. Instead of relying on manually specified rules and heuristics, our approach *uses learning classifiers to identify element labels based on form layout patterns*. The use of a learning-based approach makes our approach adaptable: to handle new form layout patterns, updated

PVLDB '08, August 23-28, 2008, Auckland, New Zealand



(a) Forms in the *Auto* domain (b) Dynamic form in the *Airfare* domain

Figure 1: Examples of web search interfaces illustrating the variability in layout design within and across different domains.

classifiers can be derived by adding forms that contain these patterns to the training set. We give an overview of the approach in Section 2.

As discussed above, there is a wide variation in the way forms are designed. Thus, an important challenge is how to construct accurate classifiers. We designed a new *hierarchical framework that performs classification in two steps* and is more effective than a monolithic classifier. The higher accuracy of the classifier ensemble is due to two main reasons: simpler classifiers can be constructed; and each classifier can use the learning technique that is best suited for its task. The classifier ensemble is described in Section 5. The process to generate the candidate mappings that serve as input to the classifier is presented in Section 3.

The classifier ensemble uses features that are automatically extracted from Web forms, as described in Section 4. Our approach extracts features that capture information contained both in the HTML markup and visual layout. The features are useful for differentiating correct and incorrect label-element mappings and include characteristics of form elements (*e.g.,* their type), characteristics of labels (*e.g.,* their font), topological information (*i.e.,* the location of the label with respect to the element—top, bottom, left, right), and the distance between the candidate labels and elements. The feature extraction process utilizes a JavaScript-aware parser and an HTML rendering engine, making it possible for the label extractor to *handle dynamic forms*.

Even within the same domain, forms may use different and conflicting layout choices (see Figure 1(a)). Consequently, the classifiers may be unable to correctly identify the label for certain elements. This may lead both to ambiguities (*i.e.,* multiple labels assigned to an element) and dangling elements (*i.e.,* elements with no associated labels). To deal with this problem, we apply a *mapping reconciliation step, where mappings obtained in the classification process are used to help identify the correct labels for form elements that remain unmatched after classification.* Previous works have observed that the set of terms used to represent attributes in form interfaces from a given domain is small [8]. The intuition behind the effectiveness of our mapping reconciliation is that terms which occur frequently in mappings across different forms are more likely to be element labels than terms that appear less frequently. Thus, when the classification fails (*e.g.,* because of a pattern that is not captured by the classifiers) and multiple or no candidate labels exist for an element, we choose the label which has the highest frequency and that is closest to the element. In Section 6, we present our mapping reconciliation algorithm.

We implemented this framework in the LABELEX (Label Extraction) system. We *performed a detailed experimental evaluation* using over 3,000 forms from a set of representative domains. The results described in Section 7 show that LABELEX is effective and able to identify labels with high precision and recall. We also *compare* LABELEX *against two state-of-the-art techniques for label extraction*: the grammar-based HSP approach proposed by Zhang et at. [23] and the heuristics-based IEXP approach proposed by He et al. [10]. LABELEX outperforms both IEXP and HSP, and it is more robust to variability in the input data.

We review related work in Section 8 and conclude in Section 9, where we outline plans for future work.

2. PROBLEM DEFINITION AND SOLUTION OVERVIEW

A Web form consists of a set of *elements* such as selection lists, textboxes, and radio buttons. Each element is usually associated with a textual label[1] which describes its meaning. For example, in Figure 1(a), the attribute *car make* is represented by label **Make:** and the attribute *car model* is represented by label **Model:**. In the remainder of this paper, we refer to element labels and form attribute names interchangeably. The set of values that can be input into a form element is referred to as the element domain. To integrate and automatically fill out Web forms, it is necessary to understand the form capabilities. This includes the ability to identify the labels and the domains for the form attributes, as well as to group related elements [23]. In this paper, we focus on the label extraction problem.

Definition 1 (Label Extraction) *Let $L = \{l_1, l_2, ..., l_m\}$ be a set of textual labels located in an HTML page P, and let W be a Web form located in P that contains set of elements $E = \{e_1, e_2, ..., e_n\}$. The label extraction problem consists of determining the set of mappings M between each form element e_i in E to exactly one label l_i in L, such that:*

$$M = \{(e_i, l_i) \mid \bigcup_{i=1}^{n} e_i = E \text{ and } \bigcup_{i=1}^{m} l_i \subseteq L \text{ and } l_i \text{ best describes the meaning of } e_i\ \}.$$

We cast the problem of label extraction as a learning task. Given a set of Web forms with labels appropriately identified, we would like to learn the patterns for label placement. Note that it is often

[1]Some forms use images, *e.g.,* GIF, to represent labels. Extracting such labels is beyond the scope of this paper.

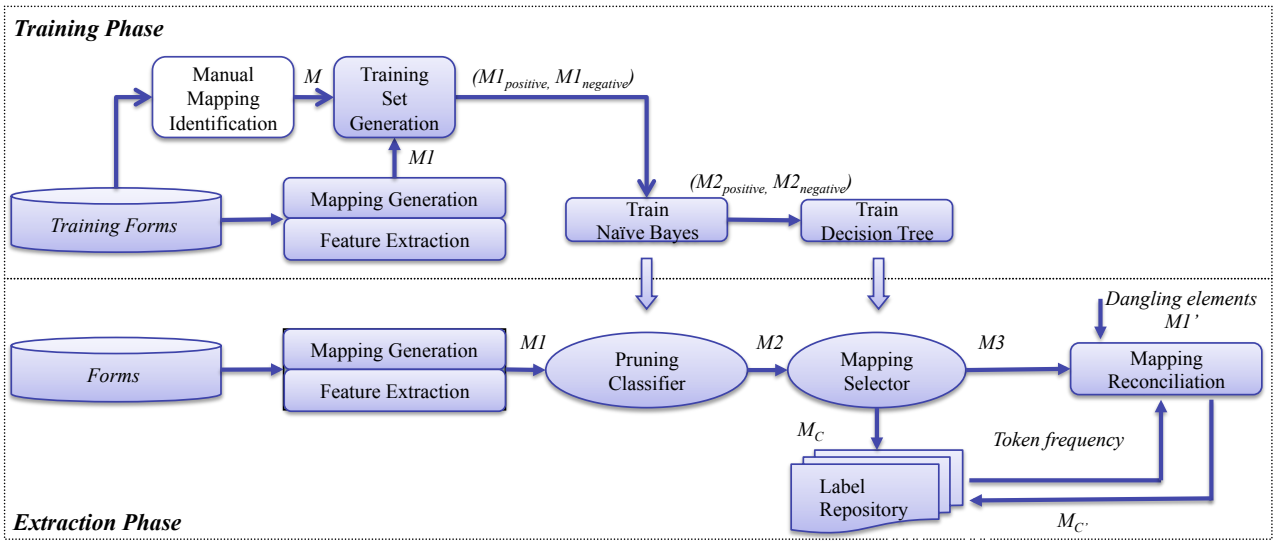


Figure 2: LABELEX: Overview.

the case that a Web page contains many more textual labels than form elements, *i.e.,* $m >> n$.

The LABELEX Approach. As illustrated in Figure 2, LABELEX operates in two phases: the training phase and the extraction phase. The *Mapping Generation* component generates a set of candidate mappings of the form:

$$m_i = (elementID, label_i)$$

where $elementID$ uniquely identifies a form element and $label_i$ corresponds to a label in the neighborhood of the element which is a possible match (Section 3). For each mapping m_i, the *Feature Extraction* module derives a set of features $featureSet_i$ associated with the label, element, and their relationship. These features include, for example, the spatial relationship between a label and an element and content similarity (Section 4).

In the *training phase*, a set of label-element mappings M is manually derived for the input forms (*Manual Mapping Identification*). These are then used to label the candidate mappings: if m_i belongs to M, then the tuple $(m_i, featureSet_i)$ is marked as a positive example; all remaining tuples are marked as negative examples. These tuples are then used to construct the first classifier (Naïve Bayes). The second classifier (Decision Tree) is trained using the output of the first classifier. A detailed description of these classifiers is given in Section 5.

The first step of the *extraction phase* coincides with that of the training phase: candidate mappings are generated and their features extracted. The set $M1$ of candidate mappings derived in the first step is used as input to the *Pruning Classifier* which removes from $M1$ mappings identified as *incorrect*. The pruned set of mappings $M2$ is sent to the *Mapping Selector*, which selects mappings classified as correct matches. The labels associated with these matches are then added to the *Label Repository*. The last step, *Mapping Reconciliation*, examines elements associated with multiple labels (*i.e.,* elements that are present in multiple mappings predicted as positive by the Decision Tree) and elements with no associated labels (*i.e.,* elements that do not belong to any mapping predicted as positive by the Decision Tree). Mapping reconciliation uses information in the *Label repository* to determine the best label for each of these elements.

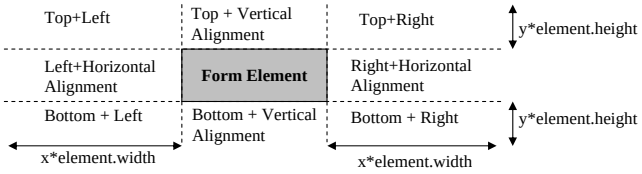


Figure 3: Spatial features of form elements and candidate positions for labels.

3. GENERATING CANDIDATE MAPPINGS

Given a Web form, LABELEX first generates a series of candidate mappings between form elements and textual labels in the neighborhood of the elements. The candidate mapping generator must identify a set of possible labels for each form element. To prevent the mapping generation to negatively impact the accuracy of the label extraction process, we need to ensure that all correct mappings are included in the candidate set. Since there are cases where form element labels are located outside the HTML `FORM` tags, we need to consider all text in the vicinity of the element.

A naïve approach would be to derive all the combinations of textual labels in the HTML page and form elements. However, this would be wasteful. As we discussed before, since forms are designed to be understood by people, the correct label for a given element is usually located in the vicinity of the element. Thus, instead of considering all possible textual labels in the Web page, as an optimization, we only consider labels that are *close* to the element.

To understand the proximity-based heuristic we use to prune the space of candidate labels, consider Figure 3. A label l is considered to be close to element e if l is located within the rectangular box $((2x+1) * element\ width, (2y+1) * element\ height)$ around e. In other words, a label is considered a candidate label if: it is located at a distance less than $y * element\ height$ from the top or bottom of e, and at a distance less than $x * element\ width$ to the left or to the right of e. We have empirically observed that this optimization is effective, and even conservative values that over-estimate the size of the bounding box (*e.g.,* $x = 3$ and $y = 5$) lead to substantial pruning while still capturing all correct mappings.

Element	Candidate labels	Correct label
Radio1	Car Only; Car+Flight+Hotel; From; To; Depart	Car Only
Radio2	Car Only; Car+Flight+Hotel; From; To; Depart; Return	Car+Flight +Hotel
Textbox1	Car Only; Car+Flight+Hotel; From; To; Depart; Return; Car pickup/drop off base on times selected; Travelers; Adults; Children; Seniors	From
Textbox2	Car Only; Car+Flight+Hotel; From; To; Depart; Return; Car pickup/drop off base on times selected; Travelers; Adults; Children; Seniors	To
Textbox3	Car Only; Car+Flight+Hotel; From; To; Depart; Return; Car pickup/drop off base on times selected; Travelers; Adults; Children; Seniors; (Max 6); (18-64) (12-17); (65+); mm/dd/yyyy	Depart
Select1	Car Only; Car+Flight+Hotel; From; To; Depart; Return; Car pickup/drop off base on times selected; Travelers; Adults; Children; Seniors; (Max 6); (18-64) (12-17); (65+); Morning	Depart

Table 1: Candidate labels for some of the elements in the *Airfare* form shown on the right of Figure 1(b).

The Candidate Mapping Generation (CMG) process works as follows. It starts by rendering the page where the form is located in a Web browser so that it can obtain information about the visual layout of elements and labels. In addition, to handle forms whose layout changes dynamically as a user interacts with it (*e.g.,* by selecting an element in a selection list), the CMG simulates user actions by invoking event handlers associated with the form elements. Then, for each element including elements rendered dynamically, the algorithm finds all candidate labels using the proximity heuristic and outputs a set of candidate mappings of the form $(elementID, label)$.

Table 1 illustrates some of the candidate mappings generated by CMG for elements in the *Airfare* form shown on the right of Figure 1(b). We can see the number of candidate labels per element is much smaller than the number of text labels located inside the form, and all correct labels are contained in candidate sets. However, the number of candidate labels per element is still large, *e.g.,* the first element has five candidate labels, and only one is correct. The large number of incorrect candidate mappings is our motivation to apply the pruning step described in Section 5.

4. EXTRACTING FEATURES

Since forms are designed for humans, intuitively, elements labels have to be placed in such way that they can be *clearly* identified. In fact, previous approaches to label extraction have observed the existence of common patterns in the layout of forms [10, 16, 23]. A label is usually located close to its corresponding element, and there are certain conventions as to where labels are placed, for example, labels for text fields are, very often, located on top or to the left of the element. Whereas in previous approaches manually-created rules and heuristics encode these patterns, our goal is to automatically *learn* these patterns and create classifiers that are able to identify the correct label for a form element.

Below, we describe the features used by the LABELEX classifiers and how they are automatically extracted. The actual classifiers are presented in Section 5. We consider different types of features, including: characteristics of form elements and labels; similarity between labels and element content; and spatial features relating labels and elements. These features are summarized in Table 2 and described in detail below.

Feature	Description
Element type	Type of form element (*e.g.,* textbox, selection list, radio button)
Font type	Font type used for label: **Bold**, *Italic* or Normal
Internal	Indicates that the label is represented as a value of the form element (*e.g.,* it is one of the values in a selection list)
Similarity	Normalized LCS score between the label and element
Alignment	Indicates whether the label and element in a mapping are horizontally or vertically aligned
Label placement	Captures the relatively position of the label with respect to the element, *i.e.,* that the label is located on top, to the left, below, or to the right of the element
Distance	Normalized spatial distance between the label and element

Table 2: List of features used in LABELEX and their meanings.

Form Elements and Labels. These features capture characteristics of elements and labels expressed in the HTML markup. We have empirically observed that some layout patterns are specific to particular form element types. For example, the label of a `checkbox` element is usually placed on the right, while the label of a `select` element is usually located on the top or on the left. To capture these element-specific patterns, for each element, LABELEX extracts its type as one of the features.

For candidate labels, we extract its font and size. These features are often a good indication of the importance of a token and are used in many applications, such as in ranking the results of a search engine [4]. For example, a token that is displayed in bold is more likely to be a label than a token in italics—Web designers tend to use italics for explanations.

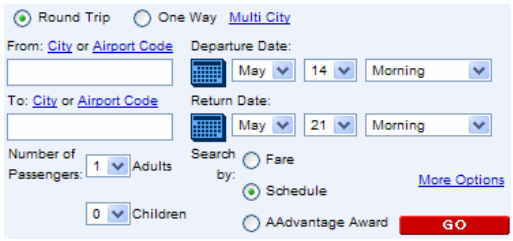
Label-Element Similarity. There are three possible ways for a Web designer to associate a textual description with a form element: by adding a label, using an internal name, or specifying the default display value of the element. Although the internal name and default display value (which can be empty) do not always provide a good description to the element, when they do, they provide good evidence for matching. To capture the string similarity between an element and candidate labels, we use normalized Longest Common Subsequence (LCS), which is defined as:

$$normalizedLCS(s_1, s_2) = \frac{LCS(s_1, s_2)}{min(length(s_1), length(s_2))} \quad (1)$$

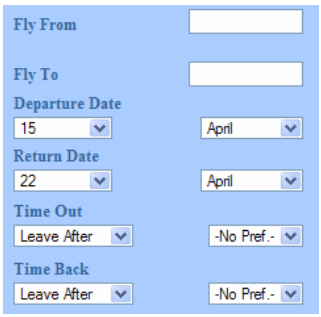
The string similarity between label and element is then computed as the minimum value of normalized LCS score between label and internal name, and between label and default display value.

Spatial Features. The spatial features exploit common design patterns. We consider both topological features and the label-element distance. Topological features capture information about horizontal and vertical alignment and they also include information about the relative position of the label with respect to the element: *top* (label is on the top of element), *bottom* (label is on the bottom of element), *left* (label is on the left of element), *right* (label is on the right of element).

In our first prototype, we used, as a distance measure, the (absolute) number of pixels between the closest edges of the corresponding bounding boxes of elements and labels. However, due to the variation in form size and density (see Figure 4), this measure was ineffective—leading to low-accuracy classifiers. A more ac-



(a)



(b)

Figure 4: Examples of (a) dense and (b) sparse forms in the *Airfare* domain.

curate (and robust) measure is obtained by normalizing the pixel distance between an element and a label by the density of the form. We define form density as a pair (*horizontal density*, *vertical density*), which measures the closeness among rows and columns in the form. The *horizontal density* is defined as

$$horizontal\ density = \frac{form\ height}{\#rows}$$

where #*rows* is the number of sets of horizontally aligned elements and labels. Similarly, the *vertical density* is defined as

$$vertical\ density = \frac{form\ width}{\#columns}$$

where #*columns* is the number of sets of vertically aligned elements and labels. If the element and label are horizontally (or vertically) aligned, the normalized distance is computed as the pixel distance between them divided by the *horizontal density* (*vertical density*). Figure 3 illustrates some of these spatial features. Our experimental results discussed in Section 7 indicate that the normalized distance is effective.

Note that, similar to candidate mapping generation, the feature extraction process also requires that the page be rendered so that both visual layout features and features of dynamically rendered elements can be extracted. In our implementation, these two steps are performed concurrently.

5. LEARNING TO IDENTIFY MAPPINGS

After the candidate mappings are generated and associated features extracted, LABELEX needs to select from the candidate set only the correct mappings. As discussed in Section 2, we cast the problem of identifying correct mappings as a learning task.

A peculiar feature of our problem is that, by construction, the set of negative examples is much larger than the positive counterpart (see Table 1). Recall from Section 3 that for each form element, a set of candidate mappings is derived. Among the mappings in this possibly large set, only one is correct. This can lead to complex and inaccurate classifiers that are likely to misclassify positive instances.

Technique	Recall	Precision	F-measure
Naïve Bayes	97.6%	30.8%	46.9%
Decision Tree (J48)	68.9%	71.4%	70%
SVM	38.1%	68.1%	48.9
Logistic Regression	54.5%	72.8%	62.4%

Table 3: Recall, precision and F-measure scores of different learning techniques for classifying candidate mappings.

Technique	Recall for Positive	Precision for Negative
Naïve Bayes	98%	92%
Decision Tree (J48)	69%	97%
SVM	38%	98%
Logistic Regression	54%	98%

Table 4: Recall and precision of different learning techniques for pruning incorrect candidate mappings.

We confirmed this experimentally. Using WEKA [18], we constructed classifiers to identify correct candidate mappings using four distinct learning techniques. The recall, precision and F-measure scores obtained by the resulting classifiers are shown in Table 3.[2] The high recall obtained by Naïve Bayes indicates that it is able to correctly identify the negative instances without removing the positive ones. But note that the precision and F-measure scores are consistently low for all classifiers.

Inspired by previous works that combine classifiers to obtain higher accuracy [3, 11], we designed a classification scheme that consists of two steps. The first classifier works as a pruning step, eliminating a large number of negative instances. The pruned set of candidate mappings, which has a better balance between positive and negative instances, allows the construction of a simpler and more accurate classifier for identifying correct mappings. The two classifiers are described below.

In order to select the *best* learning technique for each classifier, we constructed classifiers using different machine learning techniques (Naïve Bayes, J48/Decision Tree, SVM and Logistic Regression) [14] and evaluated their performance. All classifiers were built using the features described in Section 4.

Pruning Incorrect Mappings. The goal of the first classifier (the *pruning classifier*) is to reduce the number of negative instances in the candidate set. However, to ensure the overall accuracy of the classification process, it must be conservative and keep the positive instances. Hence, this classifier must have a high recall for positive instances and high precision for the negative ones. To evaluate the performance of different learning techniques, we randomly selected 5% of the forms in our dataset[3]. We set up the training phase with 10-fold cross validation and split the dataset in two thirds to the training set and one third to the test set. The average numbers for recall over positive instances and precision over negative instances are given in Table 4. These numbers indicate that the Naïve Bayes classifier is the most effective for this task.

Selecting Correct Mappings. We use the mappings produced the *pruning classifier* as the training set for the second classifier, the *mapping selector*. Using the same evaluation method as described above, we obtained the average classification accuracy for the different techniques. The F-measure scores are shown in Table 5. They indicate that Decision Tree (J48) is the most accurate for selecting the correct mappings.

[2]Formulas for computing recall, precision and F-measure are given in Section 7.

[3]Details about this dataset are given in Section 7.

Technique	F-measure for Positive	F-measure for Negative
Naïve Bayes	60%	78%
Decision Tree (J48)	79%	90%
SVM	66%	84%
Logistic Regression	64%	84%

Table 5: F-measure scores for different learning techniques for selecting correct mappings from the pruned candidate set.

Combining Classifiers. By splitting the classification process into two steps, pruning and mapping selection, it is possible to select the learning technique that best suits each problem. As we have discussed above, a Naïve Bayes classifier is the best for the pruning step, while J48 (Decision Tree) is the best for selecting the correct mappings. Our experimental results in Section 7 show that the classifier ensemble obtains high accuracy and is more effective than a monolithic classifier (see Table 3).

As Table 4 shows, the Naïve Bayes classifier is able to identify incorrect mappings with high accuracy and it also has a high recall for correct mappings. However, it is not effective at identifying the correct mappings: its precision is only 30.8% (Table 3). A limitation of the Naïve Bayes classifier comes from the fact that it assumes all features are independent, and this is not always the case. For example, if an element is a text box, its label is more likely to be located on top or to the left; while for a radio button or check box, labels are often located to the right. There are also features that are not common but that are useful for discriminating good labels. For example, although bold labels are not frequent, if a token is represented in bold, it is very likely to be a good label.

Decision Trees, in contrast, are effective at identifying mappings which are rare or that contain correlated features. The Decision Tree classifier obtains a much higher F-measure value (79%) for classifying correct mappings than Naïve Bayes (60%). The Decision Tree derived for the *Auto* domain is shown in Figure 5. Note that *Distance* is chosen as the root of the tree—it is the most discriminating feature. In fact, if a label is far from an element, for most of the cases the mapping is identified as false (see lines 22-48). If we contrast the rules in lines 21 and 47, we can see that the *Horizontal* feature reinforces *Distance* and *Similarity*: If a label and element are far apart and are not similar, the mapping is still classified as correct if they are horizontally aligned. Note that if the element and label are not horizontally aligned, the decision is false (line 37). This tree also confirms our intuition with respect to the correlations between element types and label placement: labels for radio buttons and check boxes are usually located to the right, while labels for text boxes are often located to the left (see lines 6-9).

```
01.  distance <= 0.569439
02.  |   similarity <= 0.625
03.  |   |   distance <= 0.018519
04.  |   |   |   left <= 0: true
05.  |   |   |   left > 0
06.  |   |   |   |   element_type = text: true
07.  |   |   |   |   element_type = select: true
08.  |   |   |   |   element_type = checkbox: false
09.  |   |   |   |   element_type = radio: false
10.  |   |   distance > 0.018519
11.  |   |   |   left <= 0
12.  |   |   |   |   element_type = text: false
13.  |   |   |   |   element_type = select: false
14.  |   |   |   |   element_type = checkbox: true
15.  |   |   |   |   element_type = radio: true
16.  |   |   |   left > 0
17.  |   |   |   |   vertical <= 0
18.  |   |   |   |   |   horizontal <= 0: false
19.  |   |   |   |   |   horizontal > 0: true
20.  |   |   |   |   vertical > 0: true
21.  |   similarity > 0.625: true
22.  distance > 0.569439
23.  |   horizontal <= 0
24.  |   |   similarity <= 0.7: false
25.  |   |   similarity > 0.7
26.  |   |   |   left <= 0: false
27.  |   |   |   left > 0
28.  |   |   |   |   distance <= 3.478981
29.  |   |   |   |   |   element_type = text
30.  |   |   |   |   |   |   boldFont <= 0: false
31.  |   |   |   |   |   |   boldFont > 0: true
32.  |   |   |   |   |   element_type = select
33.  |   |   |   |   |   |   boldFont <= 0: false
34.  |   |   |   |   |   |   boldFont > 0: true
35.  |   |   |   |   |   element_type = checkbox: true
36.  |   |   |   |   |   element_type = radio: false
37.  |   |   |   |   distance > 3.478981: false
38.  |   horizontal > 0
29.  |   |   distance <= 2.486742
40.  |   |   |   similarity <= 0.5
41.  |   |   |   |   similarity <= 0.444444
42.  |   |   |   |   |   similarity <= 0.428571
43.  |   |   |   |   |   |   distance <= 1.619748: false
44.  |   |   |   |   |   |   distance > 1.619748: true
45.  |   |   |   |   |   similarity > 0.428571: true
46.  |   |   |   |   similarity > 0.444444: false
47.  |   |   |   similarity > 0.5: true
48.  |   |   distance > 2.486742: false
```

Figure 5: Decision Tree (J48) in the *Auto* domain.

6. USING PRIOR KNOWLEDGE TO DISCOVER NEW LABELS

Although our classifier ensemble is able to obtain high precision and recall, it is not perfect. In some cases, it may not be able to reliably identify the label for an element, leading to ambiguous mappings (*i.e.,* multiple candidate labels for an element) and dangling elements (*i.e.,* an element with no candidate labels). There are several reasons for this. As discussed above, some forms have conflicting layout patterns. In addition, classifiers are as good as their training data and evidently, they are not able to handle patterns they have not learned.

If we consider a large set of forms in a domain, they often share a relatively small vocabulary [3, 8]. Based on this observation, LABELEX uses mappings derived by the classifier ensemble to create a label vocabulary. This vocabulary is used to reconcile ambiguous mappings and to derive new mappings for dangling elements. The intuition here is that terms with high frequency in the vocabulary are more likely to be good labels than terms with low frequency. Suppose, for example, that an unmatched element in an *Airfare* form is close to two terms: "SAVE $220" and "From". Because "From" is a term that appears frequently in this domain, it would be selected by the mapping reconciliation step.

The effectiveness of the reconciliation process is highly dependent on the quality of the constructed vocabulary, and to construct such a vocabulary, we need to identify the *important* terms. Generic terms such as "Select" are likely to have high frequency, since they often appear in composite labels (*e.g.,* "Select a City", "Select a State"). However, they are unlikely to be good labels for an element. We use a simple, yet effective strategy to select important terms. We distinguish between terms that appear alone and terms that appear in a composite candidate label. We create two separate tables: after stemming the terms that appear in the labels of successful mappings, we store all terms along with their frequencies into the *term frequency* table (TF_{table}); terms that appear in atomic

Algorithm 1 *Mapping Reconciliation* Algorithm

```
1: for each form F do
2:   for each element e_j ∈ F do
3:     if e_j is present in multiple mappings then
4:       candidateMappingSet = ⋃(e_j, l_i)
5:     else
6:       if e_j is dangling then
7:         candidateMappingSet = GenerateCandidateMapping(e_j)
8:       end if
9:     end if
10:    for each (e_j, l_i) ∈ candidateMappingSet do
11:      distance_i = compute Euclidean distance between e_j and l_i
12:      mappingscore(e_j, l_i) =
           Σ_{t∈l_j} sqrt(TF_t × STF_t) / (|l_j| × distance) + 1 / (distance + 1)
13:      correctMapping = mapping with highest score
14:      update TF_table and STF_table
15:    end for
16:  end for
17: end for
```

labels (*i.e.,* labels that consist of a single term) are placed in the *singleton frequency table* (STF_{table}) along with their frequencies.

Algorithm 1 describes the mapping reconciliation process which uses the information in these tables. It iterates over all forms and their elements. If an element is present in multiple mappings returned by the classifier ensemble, these mappings are added to the candidate set (line 3-5). For some elements, the classifier ensemble may fail to identify a label. For these dangling elements (lines 6-8), the set of candidate mappings is obtained by the Candidate Mapping Generation module (Section 3).[4] The next step is to select from the candidate set the most likely match (lines 10-15). To do so, we generate a score for each mapping according to the following formula:

$$mappingScore(e_i, l_j) = \sum_{t \in l_j} \frac{\sqrt{TF_t \times STF_t}}{|l_j| \times distance} + \frac{1}{distance + 1} \quad (2)$$

This score ensures that the (relevant) terms that have high frequency are the most likely labels, but only if they are close to the unmatched element. The denominator in the first term of Equation 2 gives a penalty to terms that are far from the element. Note that for terms that are not in the vocabulary tables (TF_{table} and STF_{table}), the score is determined solely by the distance. After the scores are calculated, the algorithm selects the mapping with the highest score.

An alternative way to leverage term frequencies would be to use them as a feature for the *Classifier Ensemble*. However, since the classifiers are trained with a small number of samples, term frequencies in the sample set may not be representative of the whole domain, in which case they may negatively affect classification accuracy. After a representative set of labels is extracted (based on the layout features), it is conceivable that the term frequencies could be effective for domain-specific classifier, but it is unlikely they it would improve the generic classifier. Furthermore, mapping reconciliation can also be useful for deriving $1:M$ mappings and possibly $N:M$ mappings (Section 9).

7. EXPERIMENTAL EVALUATION

In order to assess the effectiveness of our approach, we evaluate it against a large set of Web forms in multiple domains. We also study how the different components (i.e., the classifier ensemble and mapping reconciliation) of our solution contribute to the overall extraction accuracy. Last, but not least, we compare our approach against state-of-the-art techniques for label extraction. To the best of our knowledge, this is the first time such a comparison is performed.

[4]We can re-use the set of candidate mappings $M1$, shown in Figure 2.

Domain	FFC dataset			TEL-8 dataset	
	Forms	Elements	Sample Size	Forms	Elements
Airfare	678	7224	245	53	614
Auto	1254	7146	294	95	815
Book	747	7104	254	68	595
Movie	243	1284	149	80	645

Table 6: Number of forms and form elements from different domains for the datasets used in our experiments.

Datasets and Gold Data. We have evaluated LABELEX over four online database domains: "Airfare", "Auto", "Books" and "Movies" in two datasets. The first dataset contains 2,884 web forms automatically collected by the Form-Focused Crawler (FFC) [1, 2], a crawler that is specialized to search for forms on the Web. The second set consists of forms in the *TEL-8* dataset from the UIUC Web Integration Repository[5] and contains 296 web forms. In the remainder of the paper, we refer to the first dataset as the *FFC* dataset and to the second as the *TEL-8* dataset. Table 6 summarizes some of the characteristics of these datasets.

We should note that our datasets are representative and reflect the characteristics and layout variability of Web forms, thus, enabling us to evaluate the robustness of our approach. Table 7 shows statistics of label placement in the four domains for the FFC dataset. Note that there is a wide variation in form layout patterns both within and across the selected domains. Although labels are often aligned with their corresponding elements (see the first three columns), sometimes they are located *inside* the element. An example of an internal label is the `Select a Make` label in Figure 1(a) (right). This table also shows that the distribution of label placement in relation to form elements is different for the different domains. For example, whereas for *Airfare* and *Auto* labels are often placed to the left of elements, for *Book* and *Movie* they are most often placed to the right.

Manually-derived mappings are available in the UIUC Web Integration Repository for the TEL-8 dataset, and we used these to verify the correctness of the mappings identified by LABELEX over this dataset. For the FFC dataset, however, correct mappings are not available. Since it would not be feasible to manually extract all the labels for over 22,000 elements in this dataset, we sampled the FFC dataset to create the gold data used in our experiments. We applied the *central limit theorem* and *interval estimation theories* to approximate the original dataset distribution with a sample set [13]. We chose the confidence level as 95% and confidence interval as 5, which means we are 95% certain that the results over entire dataset are within the interval $\pm$ 5% of the results on our sample. To obtain the desired confidence, we used the sample set with sizes (*i.e.,* number of forms) shown in the fourth column of Table 6.

Effectiveness Metrics. We use the standard measures defined below to evaluate the effectiveness. Given a set of forms F, let M be the set of manually generated mappings and E the set of mappings derived by LABELEX for the forms in F:

$$Recall = \frac{|M \cap E|}{|M|} \quad (3)$$

[5]http://metaquerier.cs.uiuc.edu/repository

Domain	Right & Align	Left & Align	Top & Align	Top & Left	Internal	Others
Airfare	8%	46%	26%	1%	13%	4%
Auto	14%	52%	15%	2%	15%	2%
Book	48%	24%	1%	1%	25%	1%
Movie	44%	26%	1%	5%	21%	3%

Table 7: Label placement statistics in the FFC dataset.

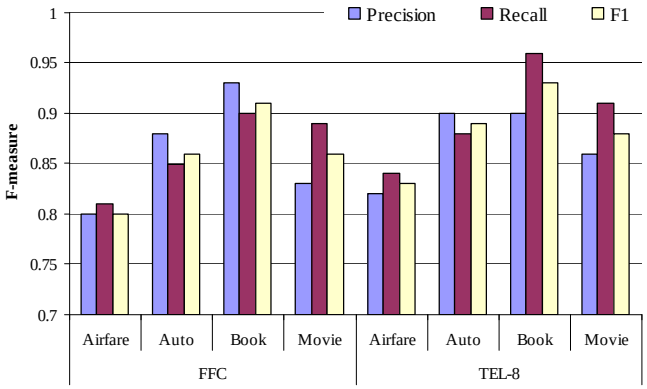


Figure 6: Precision, Recall, and F-measure scores of domain-specific classifier ensemble.

$$Precision = \frac{|M \cap E|}{|E|} \quad (4)$$

$$F-measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5)$$

Recall represents the number of correct mappings identified as fraction of all mappings considered; *precision* represents the number of correct mappings as a fraction of all mappings predicted as positive by LABELEX. The F-measure is the harmonic mean between precision and recall. A high F-measure means that both recall and precision have high values—a perfect classification would result in an F-measure with value equal 1.

7.1 Effectiveness of Label Extraction

Overall Approach. Table 8 lists the F-measure scores of the best configuration of our approach over the FFC and TEL-8 datasets. These numbers show that the combination of the classifier ensemble with the mapping reconciliation step leads to very high accuracy in label extraction. To better understand the contribution of the individual components, below we study each of them in isolation.

Classifier Ensemble. We first evaluate the performance of classifier ensembles trained for each domain separately: for each domain D, we collect a set of interfaces as the training data to build a classifier ensemble (consisting of a pruning classifier and a mapping selector) specifically for D. We refer to these ensembles as *domain-specific classifier ensembles*, or DSCE. In Section 7.2, we also consider a generic ensemble, where a single classifier ensemble is built for all domains.

Figure 6 shows the *precision*, *recall* and *F-measure* of the DSCE for the FFC and TEL-8 datasets. The F-measure scores obtained vary between 80% and 91% for the FFC dataset and between 83% and 93% for TEL-8 dataset, respectively. This indicates that the classifier ensemble is effective: it is able to automatically learn layout patterns and accurately identify label-element mappings.

Domain	FFC dataset	TEL-8 dataset
Airfare	0.88	0.89
Auto	0.94	0.95
Books	0.93	0.95
Movies	0.86	0.90

Table 8: F-measure scores for our approach over the FFC and TEL-8 datasets.

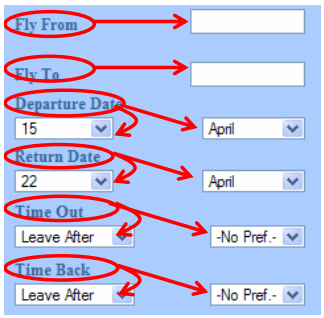


Figure 7: This form illustrates that there can be significant variation in the distance between labels and elements. It also shows labels that are related to multiple form elements.

This figure also shows that the classifier ensemble is more effective for TEL-8 than for the FFC dataset. This same pattern is observed for other approaches to label extraction (see Figure 11). The TEL-8 dataset is smaller and it was also manually collected. As a result, it is *cleaner* and presents less variability than the FFC dataset, which was automatically gathered by a Web crawler.

For the FFC dataset, the lowest F-measure score obtained by the classification process was for the *Airfare* domain (0.80), and the highest was for the *Book* domain (0.91). A possible explanation for this difference can be attributed to the layout heterogeneity in the *Airfare* domain. Examining the structure of the forms in these two domains, we observe that in *Airfare*, there is a large variation in distance between labels and elements, even within a single form. This is illustrated in the form shown in Figure 7: whereas some labels are placed very close to the corresponding elements (*e.g.,* `Departure date`), others are relatively further (*e.g.,* `Fly from`). In contrast, for *Book*, the labels are more consistently placed close to the corresponding elements. Because element-label distance is one of the features used by the classifier ensemble, it is harder for it to correctly predict labels for the more heterogeneous *Airfare* domain than for the *Book* domain.

Mapping Reconciliation. To assess the effectiveness mapping reconciliation (MR), we compute the relative improvement obtained when mapping reconciliation is applied to the mappings derived by domain-specific classifier ensemble. As Figure 8 shows, mapping reconciliation always leads to improvements, which range between 2.1% and 8.9%. These results reflect the fact that mapping reconciliation relies on the existence of labels that appear frequently across forms. Note that the accuracy boost is higher for domains that have a more homogeneous vocabulary, such as *Airfare* and *Auto*, than for heterogeneous domains (*e.g., Book* and *Movie*). Figure 9 gives some intuition for this observation: it shows the Simpson Index for the labels in the four domains we consider [17].[6] The value of this index represents the probability that two element labels selected at random from distinct forms in a domain are the same—thus, the higher the value, the more homogeneous the vocabulary is.

[6]The Simpson Index provides a better measure for vocabulary diversity than other measures such as vocabulary size.

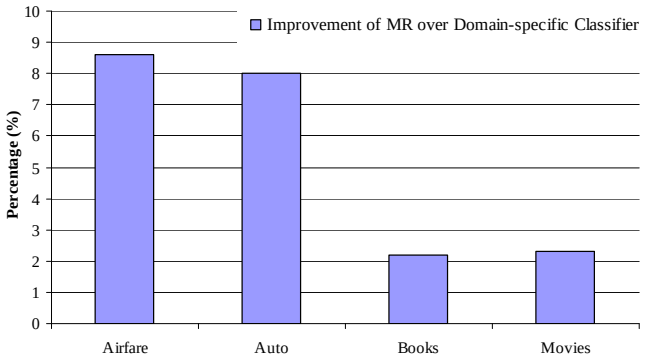


Figure 8: Improvement of mapping reconciliation (MR) over the domain-specific classifier ensemble.

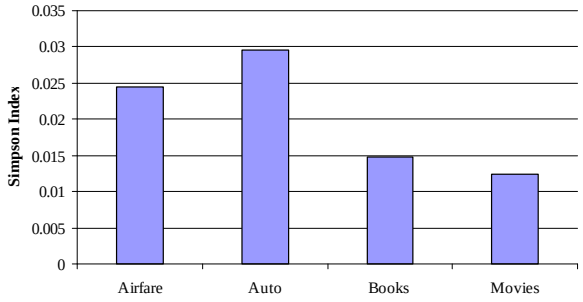


Figure 9: Simpson index for the labels in different domains. Lower values indicate higher heterogeneity.

7.2 Domain-Specific versus Generic Classifier Ensembles

The results shown in Table 8 and in Figure 6 relied on the construction of four classifier ensembles—one for each domain. An interesting question is whether it is possible to construct a single generic classifier that obtains high accuracy for forms across different domains.

Following the procedure described in Section 5, we built the *generic classifier ensemble* (GCE) as follows. First, we trained a classifier ensemble using a mix of forms from the four domains to obtain the generic classifier. The training set was generated by randomly selecting 10% forms from the sample set of each domain. In the testing phase, we ran the generic classifier and mapping reconciliation process for all the forms in each domain.

Figure 10 shows the relative improvement in F-measure score for LABELEX using the generic and the domain-specific configurations of the classifier ensembles over the TEL-8 and FFC data sets. For most domains (except for *Airfare*), the domain-specific classifier outperforms the generic one by a small margin (less than 4%). This shows that some layout patterns are indeed present in multiple domains, and that a generic classifier can be effective. Thus, the cost of training the combined classifier can be amortized by using it on multiple domains. For *Airfare*, however, the improvement is more significant, around 8%. This suggests that the choice between generic versus domain-specific depends both on the requirements of the application with respect to accuracy and on the form domains.

Note that in all four domains, the performance difference between the domain-specific and generic ensembles (without Mapping Reconciliation) is larger than that of the same ensembles combined with mapping reconciliation. That is, the improvement obtained by mapping reconciliation over the generic classifier is larger than over the domain-specific classifier. This is not surprising, since generic ensemble is more likely to fail for layout patterns that are more common for one particular domain; and in these cases, mapping reconciliation will be able to identify typical labels (*i.e.,* labels with high frequency in the domain) associated with those patterns. This reinforces the importance of mapping reconciliation and shows that it contributes to the robustness of LABELEX.

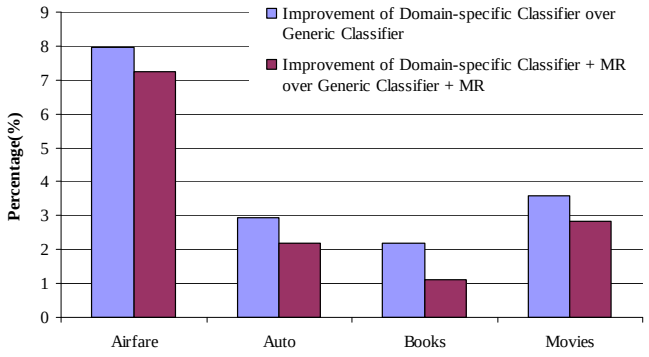


Figure 10: Generic versus domain-specific configuration. The domain-specific configuration has slightly higher accuracy than its generic counterpart.

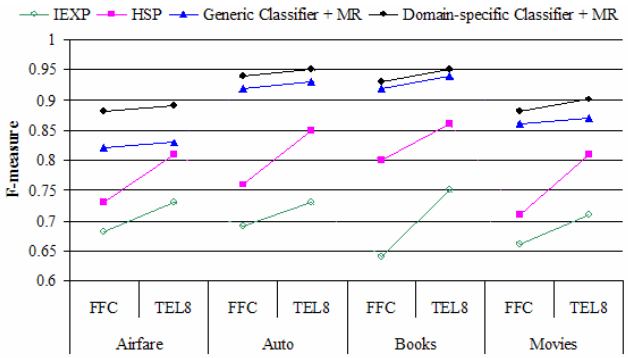


Figure 11: Comparison of our approach against HSP and IEXP. Our approach outperforms both HSP and IEXP with respect to F-measure score; and it is more robust to variability in data.

7.3 Comparison against other Approaches

To further assess the effectiveness of our learning-based approach, we compare it against the current state of the art: the grammar-based HSP approach proposed by Zhang et at. [23] and the heuristics-based IEXP approach proposed by He et al. [10]. For this set of experiments, we consider the following configurations:

- *IEXP:* our implementation of the IEXP technique following the description in [10];
- *HSP:* the binary code for HSP provided to us by the authors of [23];
- *Generic classifier + MR (GMR):* the combination of the generic ensemble and mapping reconciliation;
- *Domain-specific classifier + MR (DSMR):* the combination of the domain-specific ensemble and mapping reconciliation.

We ran these four configurations over the FFC and TEL-8 datasets. The results, shown in Figure 11, show that the two LABELEX configurations outperform both HSP and IEXP in all four domains.

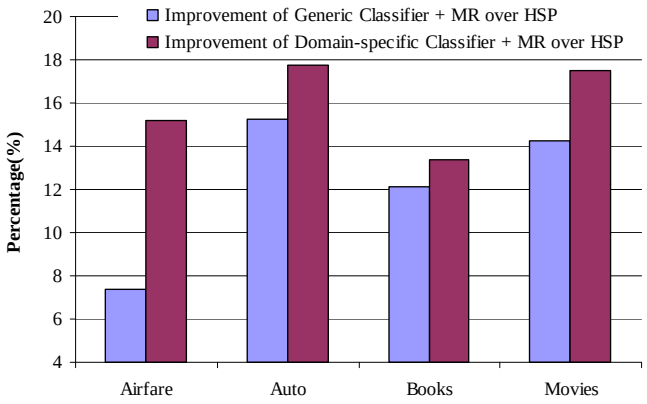


Figure 12: Relative F-measure improvement obtained by two LABELEX configurations over the Hidden Syntax Parser.

Because HSP uses a manually crafted grammar and IEXP uses a fixed set of rules, they will invariably fail for domains that do not conform with those rules, or whose layout patterns are not captured by the rules. Consider the *Book* domain. As shown in Table 7, *Book* is the domain with the largest percentage of internal labels: 25% of the FFC forms in this domain contain elements with internal labels. However, neither the IEXP heuristics nor the HSP grammar handles the case where a label is placed inside a form element. This explains, in part, why our approach performs significantly better in the *Book* domain. Note that in order to handle internal labels, all we had to do was to include examples of such elements in our training sample. In contrast, manual modifications would be required to extend IEXP and HSP to handle this case—the grammar and heuristic rules need to be updated.

The smallest accuracy difference between our approach and the others occurs in the *Airfare* domain. The IEXP heuristics and the HSP grammar work better for this domain than others, where 72% of the correct labels are placed to the left or above form elements (see Table 7).

Sensitivity to Input Data. Figure 11 also shows that the two LABELEX configurations are more robust to variations in form layout. The F-measure scores obtained by both GMR and DSMR for the FFC and TEL-8 datasets are very similar—notice the small slope angle for the lines corresponding to these configurations. In contrast, for both IEXP and HSP, the differences in the scores are much larger: they perform significantly better for the manually-collected TEL-8 dataset. Specifically, with IEXP, the difference between two datasets varies between 5.8% to 17%, and between 7.5% to 14% with HSP. While for both of our configurations, the variation is only between 1.1% to 2.2%. This fact shows that our approach is less sensitive to the data than the other approaches, or in other words, it is more robust to noise.

LABELEX vs. HSP. Let us now examine in more detail the differences between LABELEX and HSP. Figure 12 presents the relative improvement of the GMR and DSMR configurations over the grammar-based HSP. The improvement obtained by GMR varies between 7.5% and 15%, while for DSMR they range between 13% and 17.8%. Note that the smallest improvement happens for GMR over the *Airfare* domain. This is not surprising. As we noted above, the HSP grammar performs well for this domain. Furthermore, because the layout variability is high for *Airfare* and the element-label mappings are more complex, the generic ensemble is more likely to make incorrect predictions. Consider for example the form in Figure 7. Not only does it present a wide variation in the distance between labels and their corresponding elements, but it also shows that some labels can be associated with multiple elements. Because the DSMR configuration is trained with samples forms from the *Airfare* domain, it is better able to capture these features that are prevalent in this domain. This is reflected in the 15% improvement it obtains over HSP.

8. RELATED WORK

The general idea of using learning classifiers for label extraction was first described in a poster presentation [15], where we discussed preliminary results obtained with our first prototype. This paper provides a detailed description of our complete solution, including several improvements to the original prototype (*e.g.,* a more principled approach to feature extraction). Furthermore, we present a detailed experimental analysis, where we evaluate the different components of our solution and compare it against other approaches to label extraction.

Other works have addressed the problem of extracting form element labels [16, 23, 10]. The form analysis component of HiWE, the hidden-Web crawler proposed by Raghavan and Molina [16], uses a layout engine to calculate the pixel distance between the element and candidate labels. They then generate a set of candidate mappings which are ranked using a set of heuristic rules based on position, number of words, font size, etc.

Instead of relying on the visual layout of form elements, He et al. [10] examine the HTML structure of the form. They capture the textual layout of labels and elements as an interface expression (IEXP). The IEXP of a given search interface consists of three basic items: t, e and $|$, where t denotes a label/text, e denotes a form element, and $|$ denotes a row delimiter, which corresponds to HTML tags such as `<p>` and `<br>`. They define rules which determine the association between label/text and elements based on their positions in the IEXP expression.

Zhang et al. [23] cast the problem of label extraction as parsing. They hypothesized the existence of a hidden syntax and developed a soft parser and a novel grammar that allows users to declaratively specify extraction rules. These rules, like our feature set, capture both topological patterns (*e.g.,* alignment and adjacency) and proximity information. However, they do not take into account the element characteristics (*e.g.,* element type and name). They deal with the problem of ambiguities in label-element assignments as well as with incomplete grammars, by encoding not only patterns but also their precedence in the grammar. The precedence helps to resolve ambiguities by determining priorities for the grammar rules. Designing a good grammar is thus a challenging task. It requires the designer to decide which patterns to capture among a potentially large number forms, and to determine the precedence and importance levels of rules. In contrast, because LABELEX makes use of learning classifiers to automatically capture these patterns, a designer only needs to provide training data—which can be easily created using our visual interface (Section 3). An issue that can negatively impact the effectiveness of the soft parser is the presence of ambiguities. The parser must exhaustively enumerate all possible interpretations before it prunes the conflicting partial trees. Therefore, it is hard for it to achieve both accuracy and scalability because the more accurate the parser, the larger the number of rules, and the more significant the number of ambiguities and possible interpretations.

A common feature across these approaches is the fact that they rely on manually specified rules to determine element-label associations. Ours is the first work to make use of learning-based techniques to automatically infer the extraction patterns. And as we have shown in our experimental evaluation, the combination of learning classifiers with mapping reconciliation is very effec-

tive and outperforms both IEXP and the soft parser by a significant margin. Another limitation of these approaches is their inability to handle a significant number of Web forms that use client-side scripting (*e.g.,* JavaScript) to dynamically render their elements. LABELEX, in contrast, utilizes a JavaScript-aware parser and an HTML rendering engine, which enables it to extract dynamically rendered elements.

9. CONCLUSION AND FUTURE WORK

We described LABELEX, a learning-based approach for automatically extracting labels for form elements. LABELEX has two key features that set it apart from previous approaches to label extraction: it uses learning classifiers to automatically capture form layout patterns; and it introduces the idea of mapping reconciliation, which both improves the extraction accuracy and robustness in the presence of layout variability. Our experiments, over a large set of forms from different domains, have shown that LABELEX is effective and obtains high precision and recall for label extraction. In addition, LABELEX outperforms previous approaches by a significant margin.

An important advantage of LABELEX over previously-proposed heuristics and grammar-based approaches is that is does not require human input to define rules or for the selection of layout patterns. But LABELEX does require human input for training its classifiers. We should note, however, that our experience in creating the training samples using our interactive label selection tool has shown that these sample can be quickly constructed.

There are several avenues we plan to explore in future work. The mapping reconciliation module (Section 6) derives $1:M$ mappings, *i.e.,* one element can be mapped to one label, and one label can be associated to multiple elements. Our experiments have shown that this simpler approach can be effective for a large number of automatically retrieved forms. Whereas $1:1$ or $1:M$ mappings can be sufficient for some applications (*e.g.,* the categorical clustering approach proposed by He et al. [9]), for others more relaxed mappings can be desirable (or required). In future work, we intend to extend our system to derive $N:M$ mappings. This could be achieved, for example, by modifying the Mapping Reconciliation step to return the top N mappings (instead of the top mapping it currently returns).

Another limitation of our current prototype is that the classifier ensemble and mapping reconciliation modules treat each element-label mapping in isolation: they decide that a mapping is correct based solely on the features associated with the label and the element. In future work, we plan to investigate alternative strategies for mapping selection which take a more global view, *i.e.,* consider the form as a whole. Because a label may be associated with multiple elements, the straightforward approach of removing a label from the candidate set when it is used in a mapping would lead to a potentially large number of dangling elements. Thus, to improve the effectiveness of LABELEX, a global mapping generator must be able to handle $1:M$ mappings between labels and form elements.

Acknowledgments. We thank Kevin Chang and Zhen Zhang for sharing with us their implementation of HSP [23] which we used in our experimental evaluation. We also thank Eun Yong Kang for his help with the implementation of IEXP and the experimental comparison between IEXP and LABELEX. This work was funded by the National Science Foundation under grants IIS-0713637, CNS-0751152, IIS-0746500, IIS-0534628, and IIS-0513692.

10. REFERENCES

[1] L. Barbosa and J. Freire. Searching for Hidden-Web Databases. In *Proceedings of WebDB*, pages 1–6, 2005.

[2] L. Barbosa and J. Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of WWW*, pages 441–450, 2007.

[3] L. Barbosa and J. Freire. Combining classifiers to identify online databases. In *Proceedings of WWW*, pages 431–440, 2007.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of WWW*, pages 107–117, 1998.

[5] K. Chang, B. He, and Z. Zhang. Metaquerier over the deep web: Shallow integration across holistic sources. In *In Proceedings of the VLDB Workshop on Information Integration on the Web*, 2004.

[6] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large-Scale Integration: Building a MetaQuerier over Databases on the Web. In *Proceedings of CIDR*, pages 44–55, 2005.

[7] Google Base. http://base.google.com/.

[8] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In *Proceedings of ACM SIGMOD*, pages 217–228, 2003.

[9] B. He, T. Tao, and K. C.-C. Chang. Organizing structured web sources by query schemas: a clustering approach. In *Proceedings of CIKM*, pages 22–31, 2004.

[10] H. He, W. Meng, C. Yu, and Z. Wu. Automatic extraction of web search interfaces for interface schema integration. In *Proceedings of WWW*, pages 414–415, 2004.

[11] B. Heisele, T. Serre, S. Prentice, and T. Poggio. Hierarchical Classification and Feature Reduction for Fast face Detection with Support Vector Machines. *Pattern Recognition*, 36(9):2007–2017, 2003.

[12] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007.

[13] W. Mendenhall, R. J. Beaver, and B. M. Beaver. *Introduction To Probability And Statistics*. Prentice Hall, 2005.

[14] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[15] H. Nguyen, E. Y. Kang, and J. Freire. Automatically extracting form labels. In *Proceedings of ICDE*, pages 1498–1500, 2008.

[16] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *Proceedings of VLDB*, pages 129–138, 2001.

[17] E. H. Simpson. Measurement of Diversity. *Nature*, 163:688, 1949.

[18] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ ml/weka.

[19] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *Proceedings of ICDE*, pages 357–368, 2006.

[20] W. Wu, A. Doan, and C. Yu. Merging interface schemas on the deep web via clustering aggregation. In *Proceedings of ICDM*, pages 801–805, 2005.

[21] W. Wu, A. Doan, and C. Yu. WebIQ: Learning from the web to match deep-web query interfaces. In *Proceedings of ICDE*, page 44, 2006.

[22] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. In *Proceedings of ACM SIGMOD*, pages 95–106, 2004.

[23] Z. Zhang, B. He, and K. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *Proceedings of ACM SIGMOD*, pages 107–118, 2004.